Salut! Felicitări pentru inițiativă! Microserviciile sunt un domeniu fascinant și foarte relevant în dezvoltarea web modernă. Am structurat un plan de lecții, cu exerciții la finalul fiecărei secțiuni, pentru a te ajuta să înveți eficient și să-ți atingi obiectivele.

Structura Cursului: Introducere în Microservicii

Acest curs este conceput pentru a te ghida pas cu pas prin conceptele de microservicii, de la elementele fundamentale la aspecte mai avansate, dezvoltându-ți în același timp limbajul tehnic și gândirea structurată.

Lecția 1: De ce Microservicii? - O Introducere în Evoluția Arhitecturilor Software

Obiective:

- Înțelegerea problemelor arhitecturilor monolitice.
- Introducerea conceptului de microservicii ca soluție.
- Familiarizarea cu termenii de bază.
- Dezvoltarea unei perspective istorice asupra arhitecturilor software.

Conținut:

1. **Arhitectura Monolitică: Avantaje și Dezavantaje**
    - Ce este un monolit? (O singură unitate de deploy, o singură bază de cod, o singură bază de date).
    - **Avantaje:** Simplitate la început, ușor de dezvoltat și testat inițial.
    - **Dezavantaje:** Dificultăți de scalare, dificultăți de deploy, dificultăți de management al codului (complexitate crescută), dependințe strânse, dificultăți de a adopta tehnologii noi, impactul erorilor.
2. **Apariția Necesității pentru Microservicii**
    - Scalabilitatea cerută de aplicațiile moderne.
    - Echipe mari de dezvoltare și necesitatea paralelismului.
    - Flexibilitatea tehnologică.
    - Rezistența la erori (fault tolerance).
3. **Ce sunt Microserviciile?**
    - Definiție: Aplicații mici, autonome, care rulează propriile procese și comunică prin mecanisme ușoare (ex: HTTP, message brokers).
    - **Caracteristici Cheie:**
        - **Decuplare (Loose Coupling):** Serviciile sunt independente.
        - **Cohesion (High Cohesion):** Fiecare serviciu se ocupă de o singură funcționalitate sau un singur domeniu de business.
        - **Autonomie:** Echipe autonome, deploy independent.
        - **Persistență descentralizată:** Fiecare serviciu își poate avea propria bază de date.
        - **Rezistență la erori:** Defectarea unui serviciu nu afectează neapărat întregul sistem.
        - **Scalabilitate:** Se pot scala individual.

4. **Diferența dintre Monolit și Microservicii: O Viziune de Ansamblu**
    - Comparație vizuală și conceptuală.
    - Când să alegi un monolit vs. microservicii.

Exerciții (Lecția 1):

1. **Definiții:** Explică în propriile cuvinte ce înseamnă "arhitectură monolitică" și "arhitectură de microservicii".
2. **Scenariu:** Ești responsabil de dezvoltarea unei noi platforme de e-commerce. Enumeră 3 avantaje și 3 dezavantaje pe care le-ar avea o arhitectură monolitică pentru acest proiect, odată ce platforma devine populară.
3. **Gândire Critică:** O companie mică, cu o singură echipă de dezvoltatori, vrea să construiască o aplicație web simplă. Ar fi microserviciile cea mai bună alegere de arhitectură de la început? Justifică răspunsul.
4. **Analiză:** Dă exemple de companii mari (ex: Netflix, Amazon) care, după părerea ta, beneficiază enorm de pe urma arhitecturilor de microservicii și explică de ce.

Lecția 2: Principiile Fundamentale ale Designului de Microservicii

Obiective:

- Înțelegerea principiilor SOLID aplicate la microservicii.
- Cunoașterea conceptului de Bounded Context (DDD).
- Învățarea despre comunicarea inter-servicii.
- Dezvoltarea unei gândiri axate pe servicii și API-uri.

Conținut:

1. **Bounded Context (Context Delimitat) și Domain-Driven Design (DDD)**
    - Ce este un Bounded Context? Definirea granițelor unui domeniu de business.
    - Cum ajută Bounded Context la designul microserviciilor? Fiecare microserviciu corespunde unui Bounded Context.
    - Exemple: Serviciul de Comenzi, Serviciul de Produse, Serviciul de Utilizatori.
2. **Single Responsibility Principle (SRP) Aplicat la Microservicii**
    - Fiecare serviciu ar trebui să aibă o singură responsabilitate bine definită.
    - Evitarea "serviciilor gigant" sau a "serviciilor anemic".
3. **Comunicarea Inter-Servicii**
    - **Sincronă:**
        - **RESTful APIs (HTTP):** Cele mai comune. Cerere-răspuns. Avantaje și dezavantaje (latenta, disponibilitatea serviciului).
        - **gRPC:** Protocol bazat pe Protobuf, mai eficient pentru comunicații interne.
    - **Asincronă:**
        - **Message Queues/Brokers (Kafka, RabbitMQ, SQS):** Publicare/Subscriere, evenimente.
        - **Avantaje:** Decuplare puternică, rezistență la erori, scalabilitate.
        - **Dezavantaje:** Complexitate adăugată, eventuală lipsă de imediatitate.

4. **Coordonarea și Orquestrarea Serviciilor**
    - **Orchestration (Orchestrare):** Un serviciu central controlează fluxul.
    - **Choreography (Coregrafie):** Serviciile reacționează la evenimente, fără un orchestrator central.
    - Când să alegi una sau alta.

Exerciții (Lecția 2):

1. **Identificare Bounded Context:** Pentru o aplicație de gestionare a unui magazin online, identifică cel puțin 4 Bounded Contexts și descrie pe scurt responsabilitățile fiecăruia.
2. **Design API:** Imaginează-ți că ai un serviciu de "Produse" și un serviciu de "Coș de Cumpărături". Descrie cum ar putea comunica aceste două servicii folosind un API RESTful. Oferă exemple de endpoint-uri (URI-uri) și metode HTTP.
3. **Comunicare Asincronă:** Explică un scenariu în care comunicarea asincronă ar fi mai avantajoasă decât cea sincronă între două microservicii. Dă un exemplu concret.
4. **Dezbatere:** Argumentează pro și contra pentru abordarea de "Orchestrare" versus "Coregrafie" în managementul fluxurilor de date complexe între microservicii.

Lecția 3: Implementarea și Tehnologii (Introducere)

Obiective:

- Familiarizarea cu diverse limbaje de programare și framework-uri populare.
- Înțelegerea conceptelor de containere (Docker) și orchestrare (Kubernetes).
- Introducere în bazele de date poliglote.
- Dezvoltarea unei perspective asupra ecosistemului tehnologic.

Conținut:

1. **Limbaje și Framework-uri Populare pentru Microservicii**
    - **Java (Spring Boot):** Robust, ecosistem mare, matur.
    - **Node.js (Express, NestJS):** Ideal pentru API-uri rapide, eveniment-driven.
    - **Python (Flask, FastAPI, Django REST Framework):** Rapid de dezvoltat, potrivit pentru AI/ML.
    - **Go (Gin, Echo):** Performanță înaltă, concurență.
    - **C# (.NET Core):** Performanță, ecosistem Microsoft.
    - **Considerații:** Alegerea limbajului depinde de necesități, expertiza echipei.
2. **Containerizare cu Docker**
    - Ce este un container? Izolare, portabilitate.
    - De ce Docker pentru microservicii? Consistență, deploy simplificat.
    - Imagine vs. Container.
    - Dockerfile-uri de bază.
3. **Orchestrarea Containerelor cu Kubernetes (Introducere)**
    - De ce avem nevoie de orchestratori? Managementul containerelor la scară largă.
    - Ce face Kubernetes? Scalare automată, auto-healing, load balancing.

    - Noțiuni de bază: Pods, Deployments, Services. (Nu vom intra în detalii prea mari acum, doar la nivel conceptual).
4. **Baze de Date Poliglote (Polyglot Persistence)**
    - Fiecare microserviciu își poate alege cea mai potrivită bază de date.
    - **Exemple:**
        - **Relaționale (PostgreSQL, MySQL):** Pentru date structurate, tranzacții ACID.
        - **NoSQL (MongoDB, Cassandra, Redis):** Pentru date flexibile, scalabilitate orizontală, caching.
    - Avantaje și dezavantaje.
5. **API Gateway**
    - Punct unic de intrare pentru clienți.
    - Funcționalități: Routare, autentificare, autorizare, limitare de rate.
    - Exemple: Zuul, Spring Cloud Gateway, Nginx, Kong.

Exerciții (Lecția 3):

1. **Alegerea Tehnologiei:** Ai de dezvoltat un serviciu de procesare a imaginilor care va fi intens solicitat și necesită performanță ridicată. Ce limbaj de programare și framework ai alege și de ce?
2. **Dockerfile Simplu:** Scrie un Dockerfile minimal pentru o aplicație Node.js simplă (care doar afișează "Hello, Microservices!").
3. **Avantaje Kubernetes:** Enumeră cel puțin 3 avantaje majore pe care Kubernetes le aduce într-o arhitectură de microservicii comparativ cu rularea containerelor manual.
4. **Baze de Date:** Pentru un serviciu de "Recomandări" într-o platformă de streaming video, ce tip de bază de date (relațională sau NoSQL) crezi că ar fi mai potrivită pentru a stoca datele despre preferințele utilizatorilor și de ce?
5. **Rolul API Gateway:** Explică de ce un API Gateway este important într-o arhitectură de microservicii și dă un exemplu de funcționalitate pe care o poate îndeplini.

Lecția 4: Provocări și Soluții în Arhitectura de Microservicii

Obiective:

- Înțelegerea complexității operaționale a microserviciilor.
- Familiarizarea cu concepte precum observabilitate, distribuție a tranzacțiilor și securitate.
- Dezvoltarea unei mentalități proactive pentru rezolvarea problemelor.

Conținut:

1. **Complexitatea Distribuției**
    - **Distributed Transactions (Tranzacții Distribuite):** Problema asigurării consistenței datelor în multiple servicii.
        - **Saga Pattern:** Coordonarea secvențelor de tranzacții locale. (Compensating transactions).

    - **Idempotence:** Capacitatea unei operații de a produce același rezultat, indiferent de câte ori este executată.
2. **Observabilitate**
    - **Logging:** Colectarea și centralizarea logurilor din toate serviciile.
    - **Monitoring:** Monitorizarea metricilor de performanță (CPU, memorie, latență, erori).
        - **Prometheus, Grafana.**
    - **Tracing (Trace-uri Distribuite):** Urmărirea unei cereri prin multiple servicii.
        - **OpenTelemetry, Jaeger, Zipkin.**
    - Importanța observabilității pentru depanare și operare.
3. **Gestionarea Datelor (Data Management)**
    - **Eventual Consistency (Consistență Eventuală):** Datele devin consistente în timp.
    - **Data Migration:** Cum gestionezi modificările schemelor de date în servicii independente.
4. **Securitate în Microservicii**
    - **Autentificare și Autorizare:** Cum gestionezi identitatea utilizatorilor și permisiunile în servicii distribuite.
        - **JWT (JSON Web Tokens), OAuth 2.0.**
    - **Securitatea comunicării:** HTTPS, izolare la nivel de rețea.
    - **API Gateway ca punct de securitate.**
5. **Deployment și Operațiuni (Ops)**
    - **CI/CD (Continuous Integration/Continuous Deployment):** Automatizarea procesului de build, testare și deploy.
    - **Blue/Green Deployment, Canary Release:** Strategii de deploy pentru a minimiza downtime-ul și riscurile.
    - **Rollback:** Posibilitatea de a reveni la o versiune anterioară.

Exerciții (Lecția 4):

1. **Tranzacții Distribuite:** Descrie un scenariu în care "Saga Pattern" ar fi util pentru a asigura consistența datelor într-o aplicație de microservicii. Dă un exemplu concret (ex: o comandă care implică plata, inventarul și notificările).
2. **Depanare:** Ai o cerere care eșuează în sistemul tău de microservicii. Ce instrumente de observabilitate (logging, monitoring, tracing) ai folosi pentru a diagnostica problema și în ce ordine?
3. **Consistență Eventuală:** Explică ce înseamnă "consistență eventuală" și dă un exemplu dintr-o aplicație reală unde această abordare este acceptabilă (ex: numărul de "like-uri" pe o postare).
4. **Securitate API Gateway:** Cum ai folosi un API Gateway pentru a implementa autentificarea și autorizarea pentru toate microserviciile tale, fără a replica logica în fiecare serviciu?
5. **Strategie de Deploy:** Ce diferență este între "Blue/Green Deployment" și "Canary Release" și când ai alege o strategie în detrimentul celeilalte?

Lecția 5: De la Monolit la Microservicii și Arhitecturi Avansate

Obiective:

- Înțelegerea strategiilor de migrare de la monolit la microservicii.
- Explorarea unor concepte avansate.
- Consolidarea cunoștințelor și o viziune de ansamblu.

Conținut:

1. **Strategii de Migrare de la Monolit:**
    - **Strangler Fig Pattern:** Extragerea treptată a funcționalităților din monolit în microservicii.
    - **Branch by Abstraction:** Crearea de noi abstracții pentru a facilita migrarea.
    - **Anti-Corruption Layer:** Protejarea noilor servicii de complexitatea moștenită a monolitului.
    - **Migrația datelor:** Cum gestionezi datele când extragi servicii.
2. **Serverless și Function-as-a-Service (FaaS)**
    - Ce este Serverless? Nu gestionezi servere, plătești per execuție.
    - FaaS (AWS Lambda, Google Cloud Functions, Azure Functions): Eveniment-driven, scalare automată.
    - Când să le folosești în contextul microserviciilor.
3. **Service Mesh (Introducere)**
    - Ce este un Service Mesh? Un strat dedicat pentru comunicația service-to-service.
    - Funcționalități: Traffic management, securitate, observabilitate.
    - Exemple: Istio, Linkerd.
    - Când este necesar un Service Mesh? (La scară mare, complexitate crescută).
4. **Event-Driven Architectures (Arhitecturi bazate pe Evenimente)**
    - Rolul evenimentelor în decuplare și scalabilitate.
    - Event Sourcing: Stocarea tuturor modificărilor ca o secvență de evenimente.
    - CQRS (Command Query Responsibility Segregation): Separarea operațiilor de citire de cele de scriere.
5. **Refactorizarea Continuă și Mentenanța**
    - Microserviciile nu sunt o soluție magică, necesită mentenanță continuă.
    - Importanța unei culturi DevOps.

Exerciții (Lecția 5):

1. **Strangler Fig Pattern:** Explică cum ai aplica "Strangler Fig Pattern" pentru a migra o aplicație monolitică de bancă (cu funcționalități precum conturi, tranzacții, credite) la o arhitectură de microservicii. Oferă un exemplu specific.
2. **Serverless vs. Microservicii Tradiționale:** Dă un exemplu de o funcționalitate pe care ai implementa-o folosind Serverless (FaaS) și una pe care ai implementa-o ca un microserviciu tradițional, și explică de ce.
3. **Beneficiile unui Service Mesh:** Ești un arhitect într-o companie cu peste 50 de microservicii. Explică conducerii de ce ar trebui să investească într-un Service Mesh.

4. **Arhitecturi Bazate pe Evenimente:** Cum ar putea o arhitectură bazată pe evenimente (event-driven) să ajute la scalabilitatea și rezistența la erori a unei aplicații de streaming video (ex: Netflix)?
5. **Rezumare:** Pe baza a tot ce ai învățat, formulează 3 principii esențiale pe care le-ai aplica la designul oricărei arhitecturi de microservicii.

Cultura Generală Web Dev și Gândire Structurată:

Pe parcursul acestor lecții, vei observa că vei începe să:

- **Dezvolți un limbaj adaptat conceptelor:** Termeni precum "decuplare", "consistență eventuală", "bounded context", "observabilitate" vor deveni parte din vocabularul tău.
- **Îți dezvolți cultura generală raportată la web dev:** Vei înțelege evoluția arhitecturilor, de ce anumite probleme au apărut și cum au fost abordate, ceea ce te va ajuta să apreciezi mai bine tehnologiile actuale și viitoare.
- **Ai o gândire bine structurată pentru web dev:** Procesul de a descompune o aplicație mare în servicii mici, de a defini clar API-uri, de a gestiona dependențele și de a planifica pentru scalabilitate și rezistență la erori te va forța să gândești într-un mod modular și sistemic.

Pașii Următori:

După ce vei parcurge aceste lecții și vei rezolva exercițiile, vei avea o bază solidă. Pentru a merge mai departe, îți recomand:

- **Proiecte Practice:** Începe un proiect mic, personal, folosind microservicii. Poți începe cu 2-3 servicii simple și să le extinzi treptat.
- **Aprofundare Tehnologică:** Alege un limbaj și un framework (ex: Spring Boot sau Node.js cu NestJS) și aprofundează-le.
- **Cloud Providers:** Experimentează cu serviciile de microservicii oferite de platforme cloud (AWS, Azure, GCP).
- **Citit și Urmărit:** Citește cărți de specialitate (ex: "Building Microservices" de Sam Newman), urmărește prezentări la conferințe, bloguri tehnice.

Mult succes în călătoria ta de învățare! Dacă ai întrebări pe parcurs, nu ezita să le adresezi!